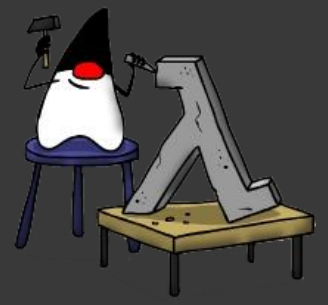
Functional Programming

Functions and Lambda Expressions


```
        this.creator = creator;
    }

    public synchronized V valueOf(K key) {
        K evict = cachePolicy.evictingKey(cache.size() == maxSize, key);
        if (evict != null) {
            cache.remove(evict);
        }
        V value = cache.get(key);
        if (value == null) {
            value = creator.create(key);
            if (value != null) {
                cache.put(key, value);
            }
        }
        return value;
    }

    public Collection<V> values() { return cache.values(); }
```



SoftUni Team

Technical Trainers

Software University

http://softuni.bg

Table of Contents

Questions

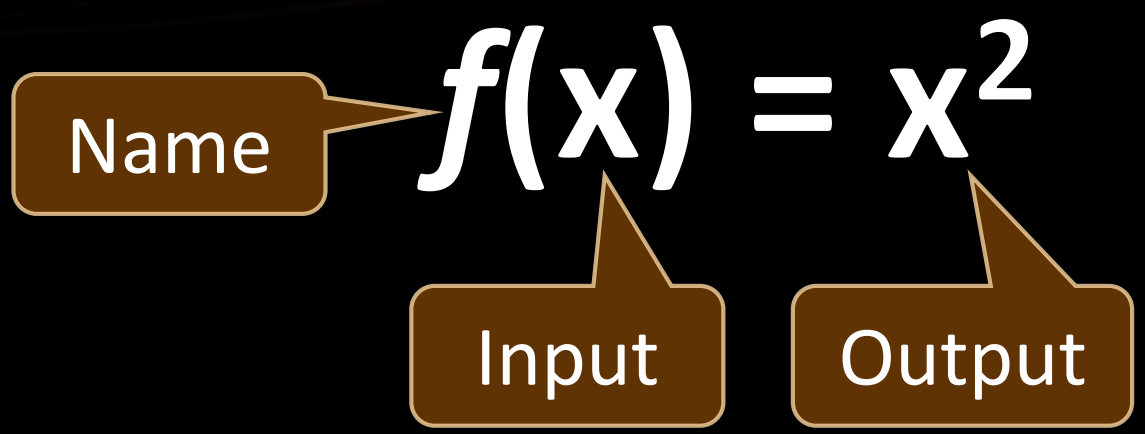
SoftUni Foundation

sli.do

#JavaFundamentals

What is a Function?

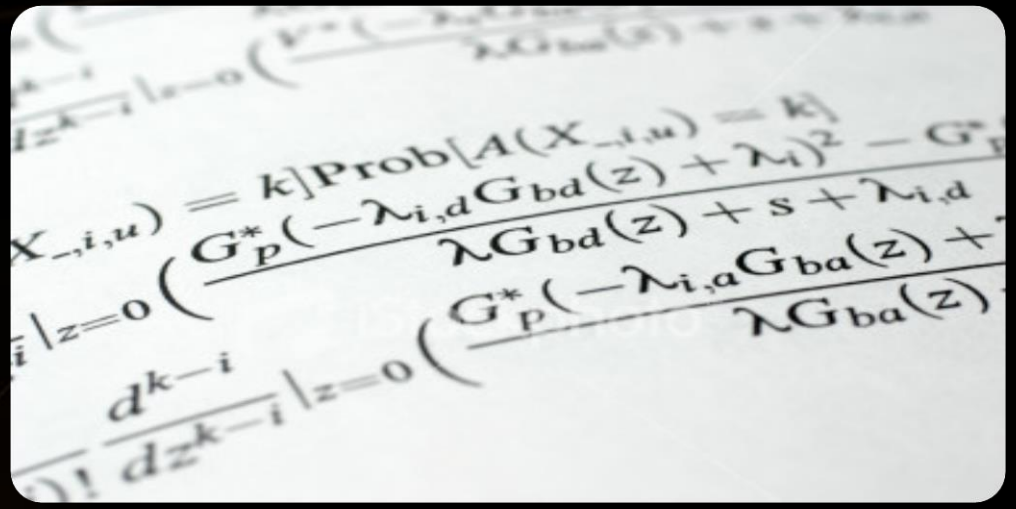
- Mathematic function



x	f(x)
3	9
1	1
0	0
4	16
-4	16

A **function** is a special relationship where **each** input has a **single** output.

SoftUni Foundation



Lambda Expressions

Lambda Expressions

SoftUni Foundation

- Lambda expression - **unnamed function**
  - Has parameters and a body

Lambda Syntax

```
(parameters) -> {body}
```

- Use the lambda operator **->** Read as "**goes to**"

Lambda Expressions (2)

SoftUni Foundation

- **Implicit** lambda expression

```
(msg) -> { System.out.println(msg); }
```

Parameters can be enclosed in **parentheses ()**

The body can be enclosed in **braces {}**

- **Explicit** lambda expression

```
String msg ->  System.out.println(msg);
```

Declares parameters' type

Lambda Expressions (3)

SoftUni Foundation

- Can have different number of parameters:
  - **Zero** parameters

```
() -> { System.out.println("Hello!"); }
() -> { System.out.println("How are you?"); }
```

  - **More** parameters

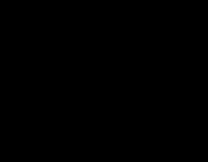
```
(int x, int y) -> { return x + y; }
(int x, int y, int z) -> { return (y - x) * z; }
```

Problem: Sort Odd Numbers

- Read Integers from the console
- **Print** the even numbers
- Sort the even numbers
- **Print** the sorted numbers

```
4, 2, 1, 3, 5, 7, 1, 4, 2, 12
```

```
4, 2, 4, 2, 12
```



```
2, 2, 4, 4, 12
```

Check your solution here: https://judge.softuni.bg/Contests/Practice/Index/463#0

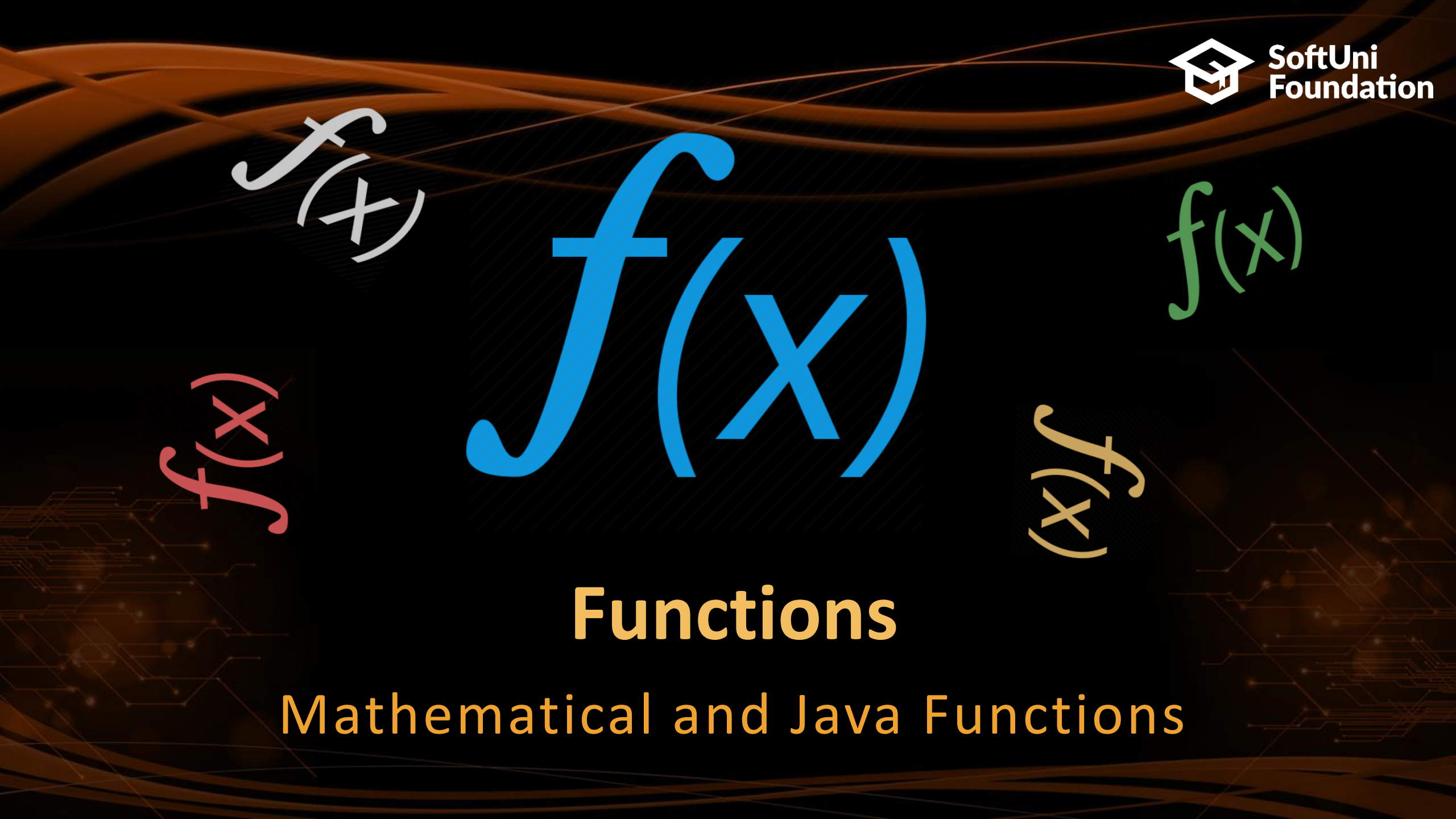
Solution: Sort Odd Numbers

SoftUni Foundation

```
//TODO: Read numbers and parse them to List Of Integers

numbers.removeIf(n -> n % 2 != 0);

// TODO: Print the even numbers

numbers.sort((a, b) -> a.compareTo(b));

//TODO: Print the sorted numbers
```

Check your solution here: https://judge.softuni.bg/Contests/Practice/Index/463#0

SoftUni Foundation



Functions

Mathematical and Java Functions

Java Functions

- In Java we can create functions
  - Analogical to mathematical functions

$$f(x) = x^2$$

```
Function<Integer, Intger> func = x -> x*x;
```

Lambda Expression

Input type

Output type

Name

Input parameter

Return expression

Function<T, R>

SoftUni Foundation

- In Java **Function<T,R>** is an interface that accepts a parameter of type **T** and returns variable of type **R**

```
int increment(int number) {
    return number + 1;
}
```

- We use function with **.apply()**

```
Function<Integer, Integer> increment = number -> number + 1;
int a = increment.apply(5);
int b = increment.apply(a);
```

Problem: Sum Numbers

- Read numbers from the console
- Print their count
- Print their sum
- Use a **Function**

```
4, 2, 1, 3, 5, 7, 1, 4, 2, 12
```

```
Count = 10
Sum = 41
```

Check your solution here: https://judge.softuni.bg/Contests/Practice/Index/463#0

Solution: Sum Numbers

SoftUni Foundation

```
Scanner scanner = new Scanner(System.in);
String[] input = scanner.nextLine().split(", ");

Function<String, Integer> parser = x -> Integer.parseInt(x);
int sum = 0;
for (String s : input) {
  sum += parser.apply(s);
}

System.out.println("Count = " + input.length);
System.out.println("Sum = " + sum);
```

Check your solution here: https://judge.softuni.bg/Contests/Practice/Index/463#0

SoftUni Foundation

Practice: Lambda Expressions

Exercises in class (Lab)

SoftUni Foundation

Other Function Types

Special Functions

Consumer<T>

- In Java **Consumer<T>** is a void interface:

```
void print(String message) {
    System.out.println(message);
}
```

- We use a Consumer with **.accept()**:

```
Consumer<String> print =
                message -> System.out.print(message);
print.accept("pesho");
```

Supplier<T>

- In Java `Supplier<T>` takes no parameters:

```
void genRandomInt()
    Random rnd = new Random();
    return rnd.nextInt(51);
```

- We use a Supplier with `.get()`:

```
Supplier<Integer> genRandomInt =
        () -> new Random().nextInt(51);
int rnd = generateRandomInt.get();
```

Predicate<T>

- In Java **Predicate<T>** evaluates a condition:

```
boolean isEven(int number) {
    return number % 2 == 0;
}
```

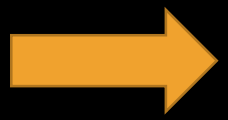
- We use the Predicate with **.test()**:

```
Predicate<Integer> isEven = number -> number % 2 == 0;
System.out.println(isEven.test(6)); //true
```

Problem: Count Uppercase Words

- Read text from the console
- Find the words starting with an Uppercase letter
- Print the count and the words
- Use a **Predicate**

```
The following example
shows how to use
Predicate
```



```
2
The
Predicate
```

Check your solution here: https://judge.softuni.bg/Contests/Practice/Index/463#0

Solution: Count Uppercase Words

SoftUni Foundation

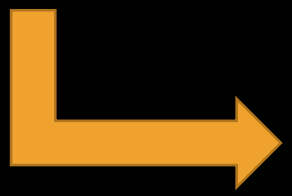
```
//TODO: Read text
Predicate<String> checkerUpperCase =
          s -> s.charAt(0) == s.toUpperCase().charAt(0);

ArrayList<String> result = new ArrayList<>();
for (int i = 0; i < textAsList.length; i++) {
  if (checkerUpperCase.test(textAsList[i])) {
    result.add(textAsList[i]);
  }
}
//TODO: Print results
```

Check your solution here: https://judge.softuni.bg/Contests/Practice/Index/463#0

Problem: Add VAT

- Read some items' prices from the console
- Add **VAT** of **20%** to all of them


```
1.38, 2.56, 4.4
```



```
Prices with VAT:
1,66
3,07
5,28
```

Check your solution here: https://judge.softuni.bg/Contests/Practice/Index/463#0

Solution: Add VAT

SoftUni Foundation

```
//TODO: Read input
List<Double> numbers = new ArrayList<>();
for (String s : input) {
  numbers.add(Double.parseDouble(s));
}

Function<Double, Double> addVat = x -> x * 1.2;

System.out.println("Prices with VAT:");
for (Double str : numbers) {
  System.out.println(String.format("%1$.2f",
                                   addVat.apply(str)));
}
```

Check your solution here: https://judge.softuni.bg/Contests/Practice/Index/463#0

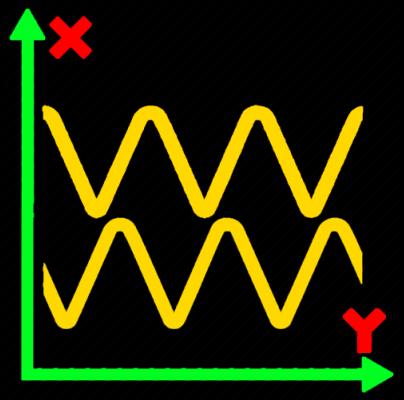
SoftUni Foundation

```
Supplier<Integer> diceRoll = () -> ThreadLocalRandom.current().nextInt( origin: 2,  bound: 12);

Consumer<String> print = message -> System.out.println(message);

for (int i = 0; i < 4; i++) {
    print.accept(diceRoll.get().toString());
}
```

Practice: Other Function Types

Exercises in class (Lab)

SoftUni Foundation



Bi Functions

Using Functions With More Parameters

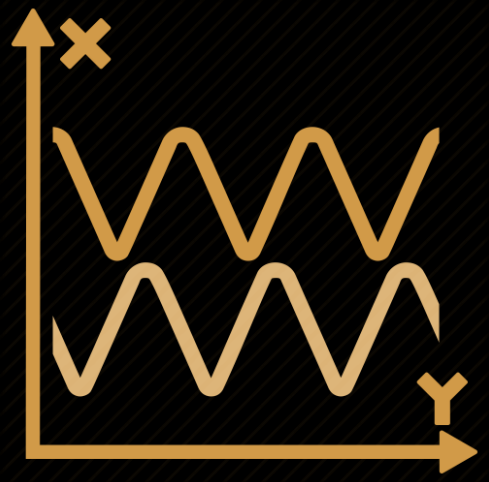
BiFunctions

- BiFunction <T, U, R>

```
BiFunction <Integer, Integer, String> sum =
  (x, y) -> "Sum is" + (x + y);
```

Two input parameters

- Analogically you can use:
  - BiConsumer <T, U>
  - BiPredicate <T, U>



Problem: Sum Numbers

- Read numbers from the console
- Print their count
- Print their sum
- Use **BiFunctions**

```
4, 2, 1, 3, 5, 7, 1, 4, 2, 12
```

```
Count = 10
Sum = 41
```

Check your solution here: https://judge.softuni.bg/Contests/Practice/Index/463#0

Solution: Sum Numbers

SoftUni Foundation

```
//TODO: Read input

BiFunction<String, String, Integer> parser =
    (x, y) -> Integer.parseInt(x) + Integer.parseInt(y);

int sum = 0;
for (int i = 0; i < input.length - 1; i + 2) {
  sum += parser.apply(input[i], input[i + 1]);
}

System.out.println("Count = " + input.length);
System.out.println("Sum = " + sum);
```

Check your solution here: https://judge.softuni.bg/Contests/Practice/Index/463#0

Passing Functions to Method

SoftUni Foundation

- We can pass `Function<T,R>` to methods:

```
int operation(int number, Function<Integer, Integer> function) {
    return function.apply(number);
}
```

- We can use the method like that:

```
int a = 5;
int b = operation(a, number -> number * 5); //b = 25
int c = operation(a, number -> number - 3); //c = 2
int d = operation(b, number -> number % 2); //d = 1
```

Problem: Filter by Age

- Read from console **n** people with their age
- Read a condition and an age so to **filter** them
- Read **format type** for the output
- Print all people that fulfill the condition

Pesho	20
Gosho	18
Radka	29
Mara	32
Izdislav	16

- Condition - "older"
- Age - 20
- Format - "name age"

Pesho	20
Radka	29
Mara	32

Solution: Filter by Age

SoftUni Foundation

```
//TODO: Read info from the console

Predicate<Integer> tester = createTester(condition, age);

Consumer<Map.Entry<String, Integer>> printer =
                                    createPrinter(format);

printFilteredStudent(people, tester, printer);
```

Check your solution here: https://judge.softuni.bg/Contests/Practice/Index/463#0

Solution: Filter by Age (2)

SoftUni Foundation

```
private static Consumer<Map.Entry<String,Integer>>
createPrinter(String format)
   switch (format)
      case "name age":
         return person -> System.out.printf("%s - %d%n",
person.getKey(), person.getValue());

private static Predicate<Integer> createTester(String
condition, Integer age)
   switch (condition)
      case "younger":
         return x -> x <= age;
```

Add more cases

Check your solution here: https://judge.softuni.bg/Contests/Practice/Index/463#0

Solution: Filter by Age (3)

SoftUni Foundation

```
public static void printFilteredStudent(
        LinkedHashMap<String, Integer> people,
        Predicate<Integer> tester,
        Consumer<Map.Entry<String, Integer>> printer) {

  for (Map.Entry<String, Integer> person : people.entrySet()) {
    if (tester.test(people.get(person.getKey()))) {
      printer.accept(person);
    }
  }
}
```

Check your solution here: https://judge.softuni.bg/Contests/Practice/Index/463#0

SoftUni Foundation

Practice: Passing Functions to Methods

Exercises in class (Lab)

Summary

- **Lambda expressions** are anonymous methods
- `Consumer<T>` is a void function
- `Supplier<T>` gets no parameters
- `Predicate<T>` evaluates a condition
- `Function<T,R>` is a function that returns `R` type
- `BiFunction<T, U, R>` accepts two parameters
- **Functions** can be passed like **variables** to methods

Functional Programming

SoftUni Foundation

Questions?

https://softuni.bg/opencourses/algorithms

Trainings @ Software University (SoftUni)

- Software University – High-Quality Education, Profession and Job for Software Developers
  - softuni.bg
- Software University Foundation
  - http://softuni.foundation/
- Software University @ Facebook
  - facebook.com/SoftwareUniversity
- Software University Forums
  - forum.softuni.bg

Software University

SoftUni Foundation

FORUM